HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Deliverable D2.1

Description of the initial version of the new front-end

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 287678.

Participant no.	Participant organisation name	Part. short name	Country
1 (Coordinator)	University of Edinburgh	UEDIN	UK
2	Aalto University	AALTO	Finland
3	University of Helsinki	UH	Finland
4	Universidad Politécnica de Madrid	UPM	Spain
5	Technical University of Cluj-Napoca	UTCN	Romania

Project reference number	FP7-287678
Proposal acronym	SIMPLE4ALL
Status and Version	Complete, proofread, ready for delivery: version 1
Deliverable title	Description of the initial version of the new front-end
Nature of the Deliverable	Report (R)
Dissemination Level	Public (PU)
This document is available from	`http://simple4all.org/publications/`
WP contributing to the deliverable	WP2
WP / Task responsible	WP2 / T2.1
Editor	Martti Vainio UH
Editor address	martti.vainio@helsinki.fi
Author(s), in alphabetical order	Juan Manuel Montero Martínez , Mircea Giurgiu, Rubén San–Segundo, Antti Suni, Martti Vainio, Oliver Watts
EC Project Officer	Pierre Paul Sondag

Abstract

On of the main goals of the SIMPLE4ALL is to replace the traditional approach to text-to-speech front-end text processing with fully data-driven approaches based on machine learning and to develop unsupervised language-independent methods for linguistic representation estimation. This report describes the initial version of the linguistic front-end of the SIMPLE4ALL system. The system for handling non-standard words, such as abbreviation, numbers and acronyms, the system for building linguistic representations in a unsupervised fashion, and an automatic prosody modelling system based on word prominences are described.

Contents

1 Introduction

Building a statistical text-to-speech synthesiser relies on large amounts of textual data and pre-recorded speech signals. Moreover, the speech signals have to be labeled according to their written form. This is usually very time consuming, and relies on manual effort from experts; it is, therefore, expensive and does not scale well to building systems for large numbers of languages. However, the hypothesis that SIMPLE4ALL is testing is that all of the methods for preparing data for TTS voice building can be automated; modern machine learning techniques that are fully data-driven can replace the expensive human labor in the process.

Replacing the traditional linguistic front-end of TTS with a fully data-driven approach based on machine learning is one of the main goals of SIMPLE4ALL . In general, this calls for a set of language-independent methods for linguistic representation estimation from data, which has itself possibly been acquired in a semi-automatic fashion from non-standard sources and/or provided by non-expert users.

The project aims to demonstrate the construction of complete speech synthesis systems starting only from speech and text, employing our novel methods for the front end in conjunction with a conventional state-clustered context-dependent HMM waveform generation module.

This report describes the first version of the new unsupervised linguistic front-end and its evaluation. There are two separate modules being described: a fully automatic text normalisation module based on machine-translation techniques and developed by UPM , and the linguistic analysis module developed by UEDIN . The evaluation of an initial system that was used to produce several voices we entered into the Spanish Albayzin Challenge is reported. Finally, a prominence based automatic prosody tagging and prediction system developed at UH is described.

The second version of the linguistic front-end is planned to include improved methods for both lexical decomposition into morph-like units and improved language modelling, as well as prosody prediction. The requirements for the current system are described in Deliverable 1.2, section 2.1.3.

2 Text normalisation

2.1 Summary

This part of the deliverable describes work in progress for developing the text normalisation module of a fully-trainable text-to-speech system. The main target is to generate a language-independent text normalisation module, based on data (instead of on expert rules) that is flexible enough to deal with all phenomena observed in the input text. We propose a general architecture based on statistical machine translation techniques, comprising three main modules: a tokenizer for splitting the text input into a token graph (tokenization), a phrase-based translation module (token translation) and a post-processing module.

2.2 Introduction

Although text-to-speech (TTS) is an area where much effort has been devoted to text normalisation, dealing with real text is a problem that also confronts other applications such as machine translation, topic spotting and speech recognition (e.g., when preparing language modelling data, or when it is necessary to associate a phoneme sequence to a written word).

In an ideal situation, text would comprise a string of fully spelled-out words, and for each word there would be an unambiguous relationship between spelling and pronunciation. But in real text, there are non-standard words: numbers, digit sequences, acronyms, abbreviations, dates, etc. The main problem for a text normalisation module is to convert Non-Standard Words (NSWs) into regular words. This problem can be seen as a translation problem between real text (including NSWs) and an idealised text where all the words are standardised.

2.3 State of the art

A good reference for current approaches to text normalisation, as used in TTS, is [1]. The authors propose a very complete taxonomy of NSWs considering 23 different classes grouped in three main types: numerical, alphabetical and miscellanea. Sproat et al describe the whole normalisation problem of NSWs, proposing several solutions for some of the problems: a good strategy for tokenising the input text, a classifier for determining the class associated to every token, some algorithms for expanding numeric and other classes that can be handled"algorithmically", and finally, supervised and unsupervised methods for designing domain-dependent abbreviation expansion modules.

Others have addressed specific problems of text normalisation too. For abbreviations and acronyms, there have been several efforts to extract them from text automatically [2, 3, 4] and to model how they are generated [5, 6].

Numbers [7] and proper names [8, 9, 10] have also been targetted. Nowadays, much effort on text normalisation is focused on SMS-like language used via mobile phones and social networks like Facebook or Twitter [11, 12, 13].

As a consequence of advances obtained in machine translation in the last decade, there has been an increasing interest on exploring machine translation capabilities for dealing with the problem of text normalisation [14, 15].

We therefore propose to use a general architecture based on statistical machine translation techniques for dealing with all challenges included in the problem of text normalisation. The main target is to generate a language-independent text normalisation module, learned from data (instead of on expert rules) that is flexible enough to deal with all situations presented in this research line. Note that we expect the model will need re-training for each new language: it is the architecture that is language-independent, not the translation model's parameters. Also, wherever we say 'language' we could also say 'domain', since domain-specific systems are one of our sub-goals.

2.4 Architecture description

Figure 2.4a shows the architecture of our proposed system. It is composed of three main modules: a pre-processing module that splits the text input into a token graph (tokenization), a phrase-based translation module (token translation) and a post-processing module for removing some tokens.

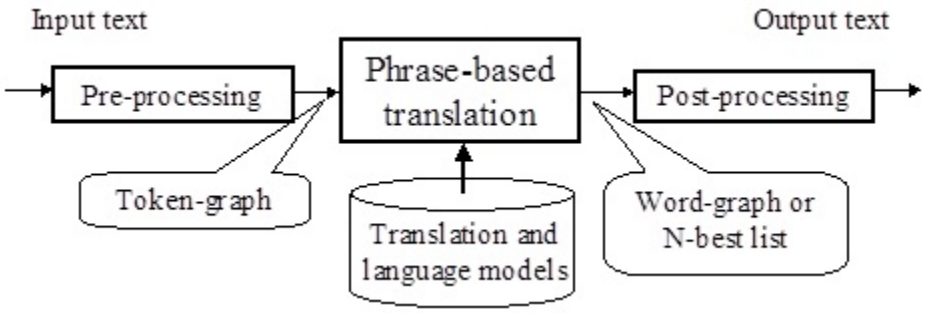

Figure 2.4a: Architecture diagram

2.4.1 Pre-processing: Sentence Tokenization

In this first module, the input text is split into tokens. This process is carried out in two steps. In the first step, a preliminary token sequence is generated considering a small set of rules. As one of the main targets of this work is to provide a language-independent architecture, the main rules should be language-independent:

- The first rule assumes that blank characters provide an initial segmentation in tokens.
- The second rule subdivides initial tokens (sequences of characters between blank characters) considering some homogeneity criterions:
  - Tokens must have only alpha or numerical characters. If there is a change from alpha to number or vice-versa, the token must be subdivided.
  - Punctuation characters must be considered as independent tokens

We are currently excluding languages that do not use blank characters to divide text into tokens, but will consider at a later stage whether to address such languages, noting that there are existing approaches that can be applied to many of these languages (e.g., Arabic) In addition to these rules, it is possible to add new rules focused on one language or on a set of languages (e.g., Romance). For example, in English, it is possible to consider subdividing a token if there is a change between lower and upper letters or vice-versa [1].

In the second step, some of the tokens are re-written in a different format in order to facilitate their subsequent translation into standard words. This starts by classifying each token as a standard word (W) or as a non-standard word (NSW). This classification can be done considering a dictionary of standard words in this language, or by considering a more complex classifier based on some features obtained from the target token and its context: character language model, vowels, capitals, etc.

If the token is classified as a NSW, it is split into letters including some separators at the beginning and at the end of the letter sequence. For example, UPM (Universidad Politcnica de Madrid in Spanish) is rewritten into # U P M #. This way of rewriting an alpha token tries to introduce a high flexibility to facilitate the text normalisation process. Considering sequences of letters, some unseen acronyms could be normalised by spelling (using the translations of its graphemes individually).

Also, all numbers are rewritten dividing the token into separated digits. Every digit is complemented with its position in the number sequence. For example: 2012 is rewritten as 2_4 0_3 1_2 2_1, where 2_4 means the digit 2 in the 4th position (beginning from the right). Roman numerals are first translated into Arabic digits and then rewritten digit by digit.

As we will see, the translation module can deal with graphs of tokens as input, not just linear strings. Thanks to this possibility, it is possible to delay hard decisions when classifying tokens as standard words or NSWs. By

representing the text as a token graph, both alternatives can be considered (with different weights if necessary). Figure 2.4b shows an example of token graph for the sentence "Welcome to UPM2012".

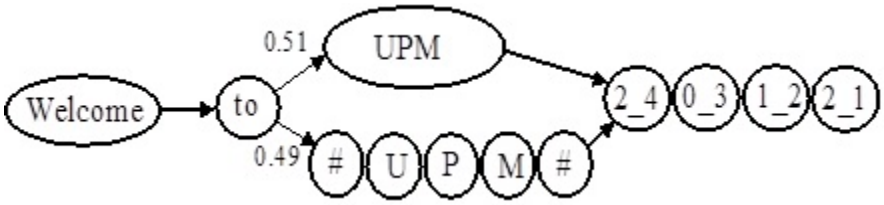

Figure 2.4b: Token graph for the sentence "Welcome to UPM2012"

The token "UPM2012" is divided into two tokens: UPM and 2012. The first one, UPM, is rewritten considering two possibilities: as it is, and letter by letter. The second one is a number and it is rewritten digit by digit, including information about its position.

The standard vs. non-standard word classifier needs to detect standard words with high accuracy, in order to reduce the token graph complexity, avoiding alternative paths in these cases.

2.4.2 Token Translation

The translation of tokens into a sequence of standard words is performed using a phrase-based machine translation system. The phrase-based translation system is based on the software released from the NAACL Workshop on Statistical Machine Translation in 2012. The translation process uses a phrase-based translation model and a target language model. These models were trained using a procedure outlined in Figure 2.4c.

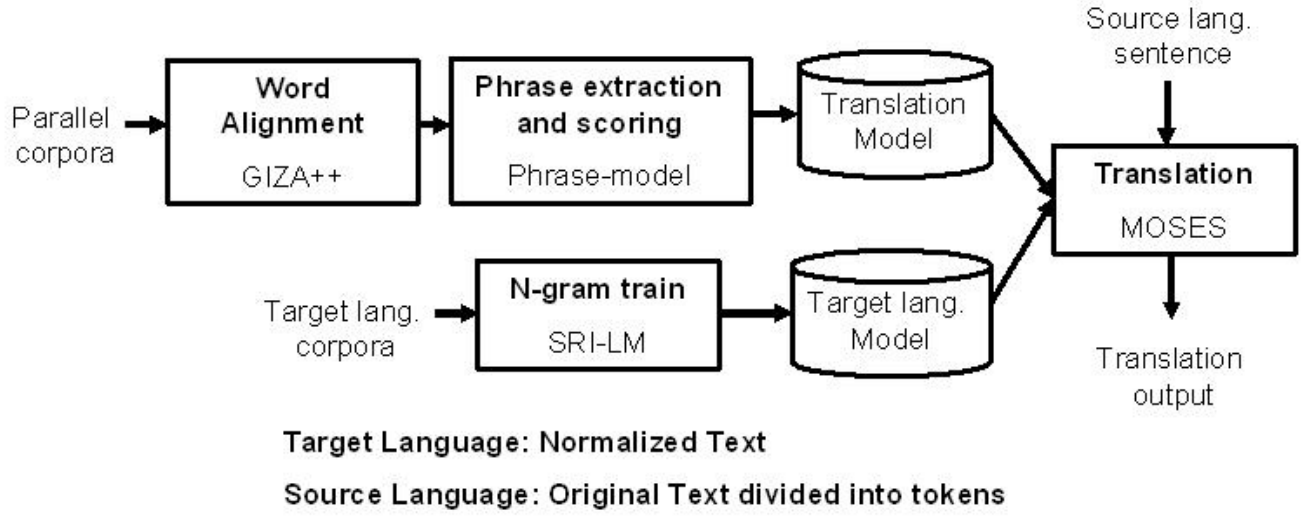

Figure 2.4c: Process for training the translation and target language models

The first step is word alignment. In this step, the GIZA++ software [16] is used to calculate alignments between source and target tokens. In order to establish these alignments, GIZA++ combines the alignments in both directions. As there are many standard words, they are the same tokens in source and target languages, so are important reference points for the alignment.

The second step is phrase extraction [17]. All token phrase pairs that are consistent with the token alignment are collected. For a phrase alignment to be consistent with the word alignment, all alignment points for rows and columns that are touched by the box have to be in the box, not outside. The maximum size of a phrase was increased to 20 in order to deal with token graphs including sequences of letters and digits properly.

Finally, the last step is phrase scoring. In this step, the translation probabilities are computed for all phrase pairs. Both translation probabilities are calculated: forward and backward.

The Moses decoder (http://www.statmt.org/moses/) is used for the translation process. This program is a beam search decoder for phrase-based statistical machine translation models. In order to build the N-gram target language model needed by Moses, the SRI language modelling toolkit was used [18].

1. Factored translation models

   Using Moses, one can train factored models in order to include this information in the translation process [19]. This possibility is an extension of phrase-based statistical machine translation models that enables the straightforward integration of additional annotations at the word or token level (linguistic markup or automatically generated word classes). The main idea is to add additional annotation at the word level. A word in this framework is not only a token, but a vector of factors that represents different levels of annotation.

   The translation of factored representations of input tokens into the factored representations of output tokens is broken up into a sequence of mapping steps that either translate input factors into output factors, or generates additional output factors from existing output factors. The information included in these factored models can be a tag with semantic information, Part-Of-Speech of a word (name, article, verb, adverb, preposition, etc.), gender or number features , verb tense, types of adverbs, etc.

   In order to increase the generalisation capability of the proposed architecture, it is possible to add, to the input token graph, an additional factor per node with information that might be useful to the normalisation process (Figure 2.4d).

   The additional factor on the source side is W for standard word, NSW for non-standard word, L for letter and N_x for digits where x is the position of the digit in the number. Including this additional factor, it is possible to increase the generalisation capability of the system.

2. Corpora necessity

   In order to learn this token translation module, it is necessary to train both a translation and a target language model. For training the translation module it is necessary to have a parallel corpus including examples of all possible types of NSWs as described in the taxonomy from [1]. Some examples are:

   - Abbreviations and Acronyms: “The UPM is ” and “The Universidad Politecnica de Madrid is ...”.
   - Numbers: “more than 120 cars” “more than one hundred and twenty cars”.
   - Dates and times: “On May 3rd, 2012” “on may third , two thousand and twelve”.
   - Webs and emails: “lapiz@die.upm.es” “lapiz at D I E dot U P M dot E S”.
   - Money and percentages: “$3.4 billions” “three point four billion dollars”.
   - Misspelling or funny spelling: “CU8er” “see you later”.

   This is the most important aspect when developing the text normalisation module. The system performance depends strongly on the data used to train the translation model. Parallel corpus generation is a costly task that should be supported with automatic procedures to reduce cost. With this idea, many efforts have been devoted to obtaining appropriate corpora from raw text with the least amount of supervision [9, 3, 7].

   One interesting characteristic of Moses is the possibility of combining different phrase tables (translation models) during the translation process using different weights. Considering this possibility, an interesting

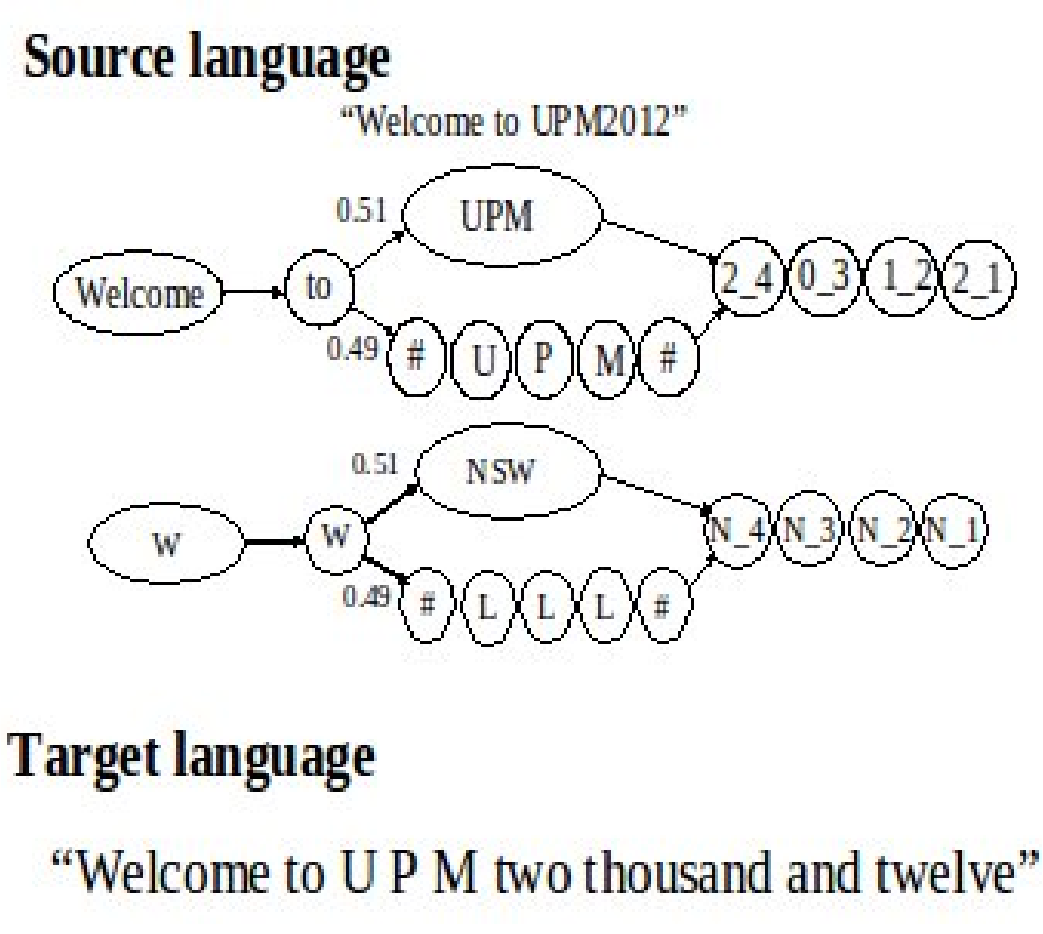

Figure 2.4d: Input token-graph with an additional factor

analysis for future work will be to compare the possibility of training individual phrase tables for each type of NSW.

One important thing to consider is that the source language (in the parallel corpora) must be pre-processed in the same way as the input text (see 2.4.1) with the difference that, in this case, the parallel corpus does not have token graphs with two alternatives but only token sequences with the correct alternative.

Regarding the target language model, it is important to use the target side of the parallel corpora, but also additional normalised sentences from different contexts in order to learn the best normalisation for a given NSW, depending on the context.

2.4.3 Post-processing

This module performs several actions in order to finally arrive at fully normalised text to be passed into the speech synthesiser. One of the main actions is to remove unnecessary tokens. For example, if after the translation module there are any # tokens remaining (which were used for defining the limits of letter sequences), they must be removed.

2.5 Experimental evaluation

The main aspects of the text normalization module that need to be tested are:

- Dealing with numbers. In this case, the main target is to define how the architecture can be adapted to deal with numerical numbers. Initially, we perform some experiments focused on isolated numbers, and then some experiments on text that includes numbers.
- Dealing with acronyms. In this case, there are two main challenges. The first one is to translate acronyms, by considering a list of known acronyms. The second challenge is to distinguish when a token is a NSW (acronym or abbreviation) or a standard word.

2.5.1 Experiments using isolated numbers

In this section, some experiments focused on numbers will be reported and some comments about the best configuration option for adapting the translation platform to deal with numbers.

2.5.1.1 Experiment 1

Description	**Amount**
Number of parallel sentences	30
Number of numbers per sentence	10
Example: 10030 20131 30232 40333 50434 60535 70636 80737 90838 90939 . diez mil treinta veinte mil ciento treinta y uno treinta mil doscientos treinta y dos cuarenta mil trescientos treinta y tres cincuenta mil cuatrocientos treinta y cuatro sesenta mil quinientos treinta y cinco setenta mil seiscientos treinta y seis ochenta mil setecientos treinta y siete noventa mil ochocientos treinta y ocho noventa mil novecientos treinta y nueve punto	

Table 2.5a: Initial database

The first experiment was carried out with the small database summarised in Table 2.5a. As we will note in section 2.5.3.4, the target language should be specified as words or in morphemes. For example, in Finnish, we suggest a morpheme based representation separating consecutive morphemes with dashes, and different words with blank characters.

Performance was measured by computing the BLEU (BiLingual Evaluation Understudy) metric and the WER (Word Error Rate). BLEU is a positive measurement (higher BLEU reveals a better system) with a limit of 100% and WER is a negative measurement (lower is better). In these very first experiments, the same set of sentences was used for training, tuning and testing. There were many problems in the alignments when training the phrase table (translation model). Table 2.5b gives the results.

System	**BLEU (%)**	**WER (%)**
Baseline (considering a grow-dial-final alignment strategy during training)	37.6	>60
Considering a tgttosrc alignment during training	65.7	55.7
Introducing a separator between consecutive numbers in the same sentence: #	85.3	22.2

Table 2.5b: Initial experiments

These problems had the following causes:

- Large difference in number of tokens between source and target languages: While having many digits in the source language (for example: 5000000 -> 5_7 0_6 0_5 0_4 0_3 0_2 0_1), in the target language there were only two words: five million. In order to solve this problem, we changed the alignment strategy, obtaining the best results by using the "tgttosrc" configuration option. With this option, the alignment for training the phrase-table is lead by the target language. As shown in Table 2.5b, we get a very good improvement when modifying the alignment method.
- Consecutive numbers caused alignment problems: To solve this, a separator was included between consecutive numbers in the same sentence, which gives another improvement in results (Table 2.5b) For example:

  - # 10030 # 20131 # 30232 # 40333 # 50434 # 60535 # 70636 # 80737 # 90838 # 90939 # .
  - # diez mil treinta # veinte mil ciento treinta y uno # treinta mil doscientos treinta y dos # cuarenta mil trescientos treinta y tres # cincuenta mil cuatrocientos treinta y cuatro # sesenta mil quinientos treinta y cinco # setenta mil seiscientos treinta y seis # ochenta mil setecientos treinta y siete # noventa mil ochocientos treinta y ocho # noventa mil novecientos treinta y nueve # punto

Of course, this experiment used a very small dataset. Our next experiment used a new database with many more training examples.

2.5.1.2 Experiment 2

Description		**Amount**
Training	Number of parallel senteces	800
	Number of numbers per sentence	1
Tuning	Number of parallel senteces	1000
	Number of numbers per sentence	1
Testing	Number of parallel senteces	4000
	Number of numbers per sentence	1
Examples: 0,105 . cero coma ciento cinco punto 201 . doscientos uno punto 504,47 . quinientos cuatro coma cuarenta y siete punto 1.496,514 . mil cuatrocientos noventa y seis coma quinientos catorce punto 71.370,656 . setenta y uno mil trescientos setenta coma seiscientos cincuenta y seis punto 1.018.448 . un milln dieciocho mil cuatrocientos cuarenta y ocho punto 88.297.519 . ochenta y ocho millones doscientos noventa y siete mil quinientos diecinueve punto 8.128.939.435 . ocho mil ciento veintiocho millones novecientos treinta y nueve mil cuatrocientos treinta y cinco punto cero coma cuatrocientos noventa y cuatro punto		

Table 2.5c: Isolated number database

The next experiments were carried out using the larger database presented in Table 2.5c. This database was been generated randomly by taking into account several patterns of numbers:

X, XXX-XXX-XXX, XXX-X.XXX, XXX-XXX.XXX, xXXX-X.XXX.XXX-XXX.XXX.XXX-XXX.XXX.XXX.XXX. All the numbers belongs to only one of the sets: training, tuning or testing.

System or experiment	**BLEU (%)**	**WER (%)**
Baseline (using tgttoserc alignment): Example: 123.400,2 -> 123 . 400 , 2 -> 1_3 2_2 3_1 . 4_3 0_2 0_1 , 2_1	80.5	10.3
Removing dots, marking millions and thousands Example: 123.400,2 -> 123400 , 2 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_1	96.1	2.2
Not dividing in different tokens using commas Example: 123.400,23 -> 123400,23 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_-1 3_-2	96.0	2.6
Not dividing in different tokens using commas Example: 123.400,23 -> 123400,23 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_-2 3_-1 In this case the codification of decimals digits starts at the end of the token	96.6	1.9

Table 2.5d: Experiments with isolated numbers

We obtain better results when considering isolated numbers for training and testing the module with a bigger database. In the baseline system in this experiment, we considered both "." and "," as token separators. So, the decimal and integer parts of a number are different tokens. Also, considering dots for separating millions, and thousands, the dots divide the number in several tokens. Analysing the errors generated by the baseline, we realised that most of the errors came from the confusion between dots referring to millions or thousands.

In order to avoid this confusion, we decided to remove dots from the numbers avoiding the system to divide the integer part in different tokens: the whole integer part is one token (see Table 2.5d for some examples). In this case, the improvement is very good, increasing the BLEU substantially, while reducing WER to close to 0%. We then decided to consider a different codification for the decimal part. But in this case, we did not get any improvement. The remaining errors are mainly of three types:

- Different ways of expressing the decimal parts: 0,04 (cero coma cero cuatro) 0,041 (cero coma cero cuarenta y uno).
  - Input: 0,91
    - Reference: cero coma noventa y uno punto
    - Output: cero coma **novecientos dieciséis** punto
- Errors when inserting the word "mil" and "millones" but not referring to a specific number:
  - Input: 1724023:
    - Reference: Un millón setecientos veintiacuatro mil veintitrés punto
    - Output: Un millón setecientos veintiacuatro mil **mil** veintitrés punto
- Errors because there is not enough training data:

- Input: 314.746.651:
  - Referente: trescientos catorce millones setecientos cuarenta y seis mil seiscientos cincuenta y uno punto
  - Output: trescientos **quince cuatro** millones setecientos cuarenta y seis mil seiscientos cincuenta y uno punto

Experiment: tuning experiments	**BLEU (%)**	**WER (%)**
Not dividing into different tokens using commas Example: 123.400,23 -> 123400,23 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_-2 3_-1 In this case the codification of decimals digits starts at the end of the token	96.6	1.9
Increasing from 6-gram to 8-gram the LM	96.7	1.8

Table 2.5e: Tuning experiments

In order to deal with the problems reported above, new experiments were run (Table 2.5e). The first change was to reorganise the codification for the decimal part. Instead of coding the distance to the comma, we tested to code the distance to the end. We got a small improvement: the other types or errors are still important. The second change was to consider an 8-gram target language model instead of the 6-gram LM used in previous experiments. We get a small improvement. In order to analyse the effect of the size of the training set, we performed 2 additional experiments increasing and reducing the amount of data available for training (Table 2.5f). It appears that good size for the training database is around 1000 examples, in order to obtain a WER lower than 2%.

Considering different amount of training data	**BLEU (%)**	**WER (%)**
400 numbers (half of the training model)	92.7	4.3
800 numbers	96.7	1.8
1800 numbers (including validation for training the models)	97.5	1.5

Table 2.5f: Experiments with different training sets

Finally, a new strategy for coding the numbers was tried and evaluated. The idea is to mix the alternatives considered previously. Initially, the number is divided in groups of 3 digits. Every group of 3 digits is coded independently (as the original proposal). This way we have less variability and more data to train. In order to deal with the problem of distinguishing “mil” and “millones” sets, specific tokens are included to help in the process. Results are given in Table 2.5g.

In order to analyse the effect of the size of the training set for this final version of the method, we performed 2 more experiments, this time reducing the amount of data available for training (Table 2.5h).

The most interesting conclusion is that we can now reduce the training set but still obtain very good results. The main types of error remaining are:

- Ambiguity like “uno mil” instead of “un mil”, or “ciento mil” instead of “cien mil”. This is the most important one.
- The second error is related to the ambiguity in several patterns. For example 1034 must be written as “mil treinta y cuarto” instead of “mil cero treinta y cuatro”. This error happens because the decimal part is coded

System or experiment	BLEU (%)	WER (%)
Not dividing in different tokens using commas Example: 123.400,23 -> 123400,23 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_-2 3_-1 In this case the codification of decimals digits starts at the end of the token	96.6	1.9
Including additional tokens Example: 123.400,2 -> 123 . 400 , 2 -> 1_3 2_2 3_1 mil 4_3 0_2 0_1 coma 2_1 In this case the codification of decimals digits starts at the end of the token	98.2	0.9

Table 2.5g: Final experiments with isolated numbers

Considering different amount of training data	BLEU (%)	WER (%)
200 numbers	97.5	1.3
400 numbers	98.2	0.9
800 numbers	98.2	0.9

Table 2.5h: Final experiments with different training sets

in a similar way than rest of 3-digit sets. This error happens only when the 3-digits set starts with 0 (10% of cases). But the error does not always appear because the language model helps with the discrimination process: on the left context there is a "comma / dot" word. It is possible to use a different codification for the decimal part, but using the same codification, we have 25% more training data for the 3-digits sets so the performance is better with less training data (fewer than 800 training examples).

2.5.1.3 Conclusions

- In order to get the best results, it was necessary to modify the tokenization process (first module) by adapting some rules to Spanish (not language independent).
- The best tokenization alternative requires to include special symbols for "mil" and "millones" recoding in groups of 3 consecutive digits. With this solution, it is possible to effectively have of more examples in the training set and it is also possible to distinguish between "mil" and "millones".

2.5.1.4 Future work

- We think it will be interesting to consider independent translation models for every NSW. We believe this solution will be necessary to adapt the translation model configuration to every NSW type.
- To evaluate other codification strategies or using factored models.

2.5.2 Experiments using isolated dates

In this section, some experiments focused on dates will be reported and some comments given about the best configuration option for adapting the translation platform to deal with numbers.

The database has been generated randomly by taking into account several patterns of dates: XX/XX/XX,XX/XX/XXXX,XX/sep/XX,XX-XX-XX,XX-XX-XXXX and XX-sep-XX. All the dates belongs to only one of the sets: training (900 dates), tuning (1000 dates) or testing (4000 dates).

System or experiment	BLEU (%)	WER (%)
Baseline (using tgttoserc alignment):	99.7	0.1

Table 2.5i: Experiments with isolated dates

The experiments produced very good results, so we conclude that this problem is relatively easy. The few errors detected are in fact errors in the translation of some numbers.

2.5.3 Preliminary experiments with acronyms

The experiments for acronyms were carried out in two phases. In the first phase, we performed some experiments considering this as a translation problem in the same was as for numbers and dates. Initially, we did not consider any modifications of the tokenization step. Only the word alternative (see section 2.4.1) is considered (the token is not rewritten into characters). All the knowledge is included in the translation module. In the second phase, we included some additional knowledge in the tokenization process: both graph alternatives (word and spelling) are considered and they have different graph scores depending on certain information.

2.5.3.1 Including knowledge only at the translation module

For these experiments, we used the database described in Table 2.5j. The division into training/tuning/testing sets was made randomly.

Description		Amount
Training	Number of parallel sentences	4054
	Number of acronyms per sentence	1
Tuning	Number of parallel sentences	500
	Number of acronyms per sentence	1
Testing	Number of parallel sentences	671
	Number of acronyms per sentence	1
Examples:		
La OTAN lanzar un .	La OTAN lanzar un punto	
PNV ETA ha empezado .	pe ene ube ETA ha empezado punto	
de PSV reventaran los .	de pe ese ube reventaran los punto	
del PNV , Joseba .	del pe ene ube , Joseba punto	

Table 2.5j: Database description for experiments with acronyms

In the training set, we included examples of the correspondence (in spoken Spanish) for every character in both standard and capital letters: for example: F (efe) , f (efe). As in the experiments with numbers, the performance of the text normalization module was measured by computing the BLEU (BiLingual Evaluation Understudy) metric and the WER (Word Error Rate). All these experiments share the same training, tuning and testing sets of sentences. Table 2.5k presents the results for this experiment.

The errors mainly arise from three causes. The first one is of course acronyms that do not appear in the training set (**12 cases**):

- Example: Original: SLMS) , .
  - Reference: “ese èle ème ese) , punto”

Initial Experiments	BLEU (%)	WER (%)
Baseline	96.1	2.9
Including 800 numbers for training the translation model	95.8	2.8

Table 2.5k: Initial Experiments

- Output: “SLMS) , punto” (leave the acronym as it is)

Secondly, we found that some times or dates are not correctly handled (**2 cases**).

- Example: Original: 21:37 GMT, pero.
  - Reference: “veintiuna horas y treinta y siete minutos ge ème te , pero punto”
  - Output: “veintiuno : treinta y siete minutos ge ème te , pero punto” (not treated yet)

The third type of error is related to some numbers that the acronym module does not translate properly. The problem is caused by numbers that do not appear in the training data (**3 cases**):

- Example: Source language: ESP) 1.144 .
  - Reference: e ese pe) mil ciento cuarenta y cuatro punto
  - Output: e ese pe) mil uno cuatro cuatro punto

Finally, there are other types of errors such as insertions or other types of processing required (**3 cases**):

- Example: la CEOE << punto
  - Reference: la CEOE menor que menor que punto
  - Output: la CEOE << punto

In order to isolate the influence of the errors in numbers, we repeated the experiment but added to the training set the training set from the earlier number experiments. We only obtained a very small improvement because the number of errors due to numbers is not very high. Also, the numbers in this test set appear in a context and in the training set they appear alone, so it would be necessary to train a better language model that covered more numbers in different contexts. These differences depend also on how the tuning process trains the weights, but the differences are not significant.

2.5.3.2 Delaying the hard decision whether a word is a standard word or an acronym

Experiments	Word	Spelling	BLEU (%)	WER (%)
Baseline (without numbers)	1.00	0.00 (disable)	96.1	2.9
More probability to word	0.99	0.01	42.6	40.2
Same probabilities	0.50	0.50	15.7	67.4
More probability to the spelling	0.01	0.99	15.0	67.8

Table 2.5l: Experiments with different costs

Using the same database as the previous section, we conducted several experiments with different weights for each graph alternative (Table 2.5l). With the baseline system, we obtained the same results as presented in Table 2.5k. When including the possibility of spelling out a word, the error is increased a lot. This increase is due to the fact that when a word is not in the training the best alternative is to spell, including any actual words. For example: for the source sentence "la ONU, Occidente ."

- Reference: "la ONU, Occidente punto"
- Output: "la ONU, o ce ce i de e ne te e punto": "ONU" is in the training set but "Occidente" not

To consider further these types of error, we decided to start introducing some knowledge in the tokenization process. The first idea was to allow spelling only in those tokens that do not appear in the source language training set. It means that we allow spelling only for those tokens not used to train the translation model. If the token is already in the training set, it means that the translation model can have learnt how to translate it. It is possible to consider a threshold in the number of appearances. As it is shown in Table 2.5m, the results improve but there is still an important difference with the baseline. What we saw is that when spelling is permitted, this option is considered in most of the cases independently of the weight (the system prefers spelling than an unknown token).

Experiments	**Word**	**Spelling**	**BLEU (%)**	**WER (%)**
Baseline (without numbers)	1.00	0.00 (disable)	96.1	2.9
More probability to word	0.99	0.01	55.6	29.8
Same probabilities	0.50	0.50	56.3	29.8

Table 2.5m: Experiments with different costs and not spelling for tokens in the training set

In order to increase the number of constraints for spelling a word, we considered the perplexity of the sequence of characters, computed using a language model trained with the character sequences of the target language training set. We used a 6-gram language model. If the perplexity is lower than a threshold, then the word is not spelled: it means that the character sequence is similar to previously-observed common sequences in Spanish and the word should therefore be pronounced directly without spelling. Table 2.5n shows the performance for different perplexity thresholds. (**Note: pronouncing an acronym as a standard word instead of spelling it, it is an alternative valid in Spanish but perhaps not in every language.**)

Experiments	**Word**	**Spelling**	**BLEU (%)**	**WER (%)**
Baseline (without numbers)	1.00	0.00 (disable)	96.1	2.9
PP threshold = 50	0.50	0.50	92.2	5.6
PP threshold = 100	0.50	0.50	95.7	3.5
PP threshold = 250	0.50	0.50	96.0	3.0
PP threshold = 500	0.50	0.50	96.2	2.8
PP threshold = 1000	0.50	0.50	96.1	2.9
PP threshold = 10000	0.50	0.50	96.1	2.9
PP threshold = 100000	0.50	0.50	96.1	2.9

Table 2.5n: Experiments with different PP thresholds

Analysing the perplexity threshold, we see that there is a value at which we can improve slightly the results. This improvement is very small. We can analyse possible causes. The 12 cases of acronyms that do not appear in the training set are given, with their perplexity, in Table 2.5o.

CSD 36.63	BCCI 29.05	WASP 55.60
MNAC 25.00	AMX 513.59	MCB 40.74
SLMS 52.17	IND 30.00	SCMM 40.39
TMC 22.24	Javier 11.64	CJC 66.07

Table 2.5o: Acronyms that do not appear in the training set, with their perplexity

Analysing the data, we see that there are several errors. For example, "Javier" appears as a spelled name but it should be considered a standard word. Also in the training set, there are several cases like UGT or RTVE that appear as standard words. Their translation is the same word. These examples also introduce errors in the language model estimation process.

One interesting example is IND. IND is part of a word in Spanish so it has a low perplexity. In this case, the algorithm should also consider a mark for beginning and end of the character sequence: <s> I N D </s>. This way, it is possible to distinguish if it is a whole word or a part of a word. In Table 2.5p, we present the results for different PP thresholds. We do not get much improvement but we think the model is better.

Experiments	**Word**	**Spelling**	**BLEU (%)**	**WER (%)**
Baseline (without numbers)	1.00	0.00 (disable)	96.1	2.9
PP threshold = 50	0.50	0.50	94.9	2.9
PP threshold = 100	0.50	0.50	95.8	3.3
PP threshold = 250	0.50	0.50	96.2	2.8
PP threshold = 500	0.50	0.50	96.2	2.8
PP threshold = 1000	0.50	0.50	96.2	2.8
PP threshold = 10000	0.50	0.50	96.1	2.9
PP threshold = 100000	0.50	0.50	96.1	2.9

Table 2.5p: Experiments with different PP thresholds considering end and begin marks

Analysing the results, we also realized that considering the perplexity of the whole character sequence may not be a good feature. In some cases, part of the character sequence is very frequent in Spanish. For example: MNAC, NAC has a very high probability and only MN generates a low probability. Considering the whole perplexity we miss the details. One alternative could be to consider the probability character by character and to compute the lowest probability, the average and the standard deviation. Table 2.5q shows experiments considering the lowest probability

This measure (lowest probability) could be useful but it depends on how well the character language model has been trained. This measurement is more unstable than perplexity, so it needs a better character language model.

Analysing in more detail both measurements, we realized that for this test set, using the perplexity sorts the tokens better. But there are still several aspects that the system can not deal with. The most important aspect is that with high PP, we find acronyms but also foreign words (English words) that must be translated as they are (based on the test reference) while the system spells them. For further analysis, it would be necessary to have a database with acronyms in order to analyse in detail the best features for discriminating the best sequence.

2.5.3.3 Analysis of features based on a character language model

In this analysis, we considered several features based on a character language model in order to detect if an acronym must be spelled or pronounced as a standard word. We have considered a database with 1340 acronyms: 685 must be spelled (SPE set) and 654 can be pronounced as standard words (SW). Considering the SW set,

Experiments	Word	Spelling	BLEU (%)	WER (%)
Baseline (without numbers)	1.00	0.00 (disable)	96.1	2.9
Lowest probability less than = -2.0	0.50	0.50	57.3	29.1
Lowest probability less than = -1.0	0.50	0.50	53.3	29.1
Lowest probability less than = -5.0	0.50	0.50	95.5	3.8
Lowest probability less than = -6.0	0.50	0.50	96.1	3.0
Lowest probability less than = -7.5	0.50	0.50	96.2	2.8
Lowest probability less than = -10.0	0.50	0.50	96.2	2.8
Lowest probability less than = -15.0	0.50	0.50	96.2	2.8
Lowest probability less than = -100.0	0.50	0.50	96.1	2.9

Table 2.5q: Experiments with different lowest probability thresholds considering end and begin marks

we trained a 6-gram character language model. For each acronym, we have computed several features trying to discriminate between both sets. The analysed features are:

- Character Perplexity: considering the character language model what is the perplexity of the character sequence, given an acronym. It is expected that SW set acronyms must have lower perplexity than SPE set acronyms
- Minimum probability: the minimum probability computed along the character sequence.
- Maximum probability: the maximum probability computed along the character sequence.
- Average N-gram: This measurement is the average order of N-gram for computing the probability of every character.
- Minimum N-gram: the minimum order considered to compute the probability of one of the character in the acronym
- Maximum N-gram: the maximum order considered to compute the probability of one of the character in the acronym.

Figure 2.5a shows ROC curves for all the features. As can be seen, the best features are the character perplexity and average N-gram order. In both cases, it is possible to reach a 10% EER (Equal Error Rate). When combining all the features it is possible to reduce the EER to 8.5%. We have obtained the best classification result by training a J48 decision tree.

2.5.3.4 Conclusions

In the future, it will be possible to extend this analysis and to consider other heuristic measurements including features from contextual tokens or even the information of a token language model. Also, it is interesting to consider the length of the token (long tokens tend not to be acronyms) and the use of capital characters. These two aspects are probably very important for distinguishing acronyms from foreign words, for example.

When using a character language model, it is very important to improve this character language model. In these experiments, we only considered 3500 words (from the target language training set) with only 1 appearance per word. One idea to improve the character language model is to include a very large database from a dictionary and/or to consider the word frequency in the estimation of the probability of the character sequence.

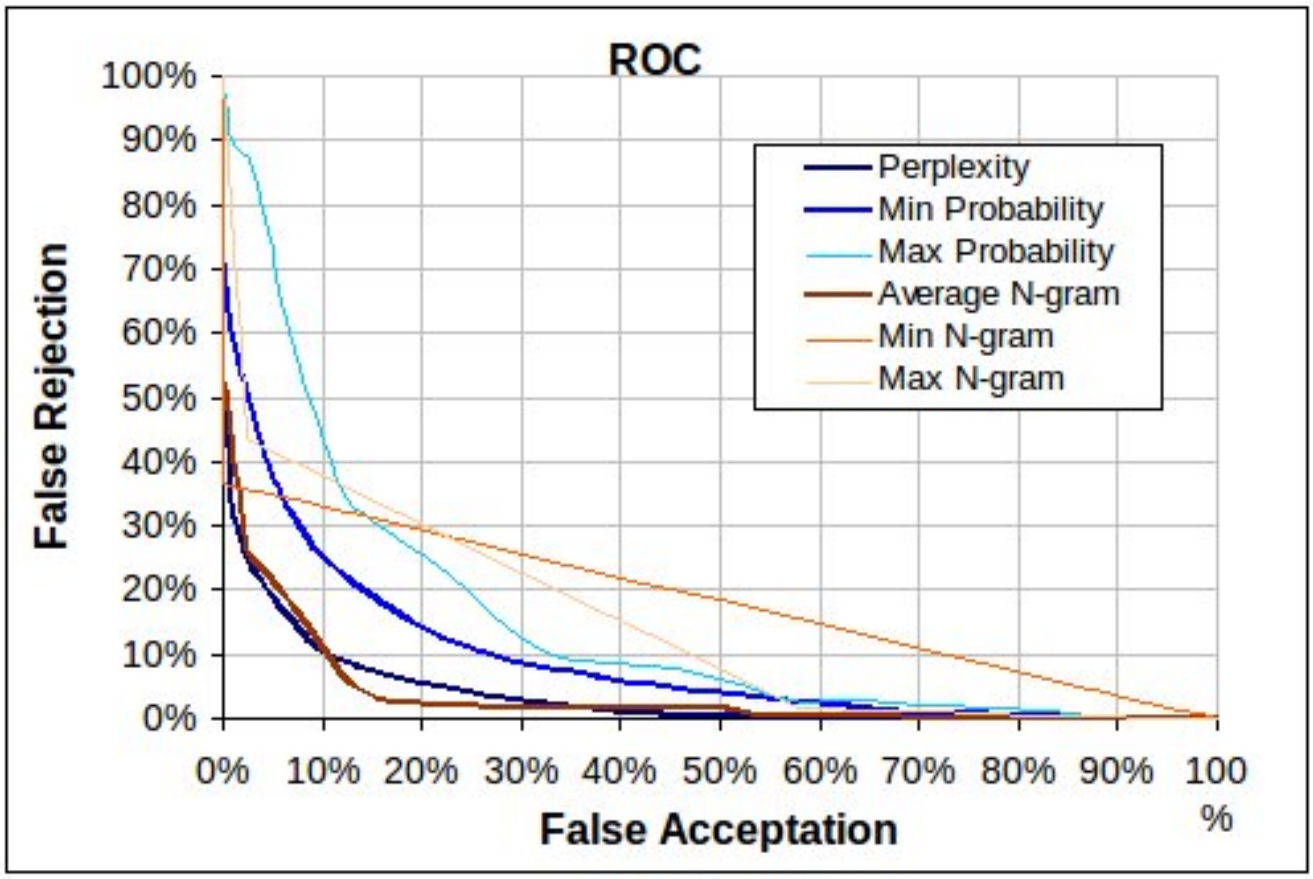

Figure 2.5a: ROC curves for the different features

2.5.4 Experiments with sentences

After analysing isolated numbers and acronyms, we now turn to some preliminary experiments on real sentences from newspapers. We obtained 2100 sentences from the web and divided them randomly into training (80%), validation (10%) and testing (10%) sets. In the training set, we also include the training sets for the numbers and acronyms.

Taking into account the isolated acronym classification results described in Section 2.5.3.3, and using the same features introduced there, we trained a J48 decision tree classifier for detecting acronyms in sentences. Initial testing gave WER=15% and BLEU=87%. These results need to be improved in future experiments. We have already detected several errors that must be addressed next:

- The sentences include dates and times that are not processed in this approach.
- Certain punctuation marks like -,:; and so on must be addressed properly.
- There is confusion between the tokenization process for numbers and several punctuation marks. For example, the comma: the number 0,34 is coded as 0_1 , 3_-1) 4_-2). The comma should probably be coded in a different way than the standard comma for pauses.

After correcting some of the errors mentioned previously, by using a new strategy to code numbers (see 2.5.1) and by including more data in the training set, such as isolated letters, a round of experiments under different conditions were made in order to discover whether the changes were successful.

There were two main groups during the experimentation: one including dates in all the sets (training, tuning and testing) and another with no dates at all. In both cases, the experiments were made including only upper case letters and including both upper and lower case letters in the training set. This will help pinpoint the possible differences between the various methods of training.

To do the experiments, the 2000+ sentences were divided into eight groups of similar size. Six of them include acronyms, numbers, letters and, if necessary, dates training sets, becoming a complete training set. One will be used to tune the system and the last one will test it. This process will be repeated until all the parts of the sentences are used to train, tune and test (i.e., a round-robin design), contrasting results.

Finally, as shown in 2.5r and 2.5s, it can be seen that there is not any remarkable difference between the two main groups of experiments. An important conclusion may be that the inclusion of dates does not deteriorate the system.

	Upper and lower case	
Round	**BLEU (%)**	**WER (%)**
1	96.40	2.45
2	96.23	2.92
3	95.68	3.14
4	96.08	2.46
5	90.85	5.57
6	94.21	4.04
7	96.25	2.62
8	96.27	2.64
Average	95.25	3.23

Table 2.5r: Results obtained from the experiments not including dates

	Upper and lower case	
Round	**BLEU (%)**	**WER (%)**
1	96.66	1.97
2	96.18	2.46
3	95.87	2.64
4	96.55	1.94
5	92.65	4.29
6	92.99	4.77
7	96.63	2.03
8	96.33	2.27
Average	95.48	2.80

Table 2.5s: Results obtained from the experiments including dates

Some of the mistakes made by the system are illustrated in these examples:

- El Schalke se complica la Champions veintidós de abril del dos mil doce punto
  El Schalke se complica la Champions veintiós de abril **de** dos mil doce punto

- Con respuesta al acontecimiento del treinta de junio de mil novecientos sesenta se han reunido cincuenta mil ochocientos cincuenta y cinco personas punto
  Con respuesta al acontecimiento del treinta mil del **j_2 u_1 de** de mil novecientos sesenta se han reunido cincuenta mil ochocientos cincuenta y cinco personas punto
- Partido Comunista del Pueblo Andaluz (pe ce pe a)
  Partido Comunista del Pueblo Andaluz **(PCPA)**
- Declaro el veintiocho de diciembre del trece como el día de la hamburguesa punto
  Declaro el **veintinueve barra doce barra trece** como el día de la hamburguesa punto
- Morirás el trece de enero del dos mil seiscientos cuarenta y tres punto
  Morirás el trece **mil del e junio de** del dos mil seiscientos cuarenta y tres punto

The errors are marked in bold. Some of these mistakes are fairly common, such as the first one, where the word "del" is translated into "de" because both words exist and "de" is a highly frequent word in Spanish. Other common mistakes are related to the new coding used in the numbers translation and spelling pronounceable acronyms or not spelling the non-pronounceable ones.

2.6 Expanding to other languages

2.6.1 Data required to create learnable text normalisation for any language

In order to perform some of these experiments with other languages, and to gauge the eventual requirements when working with a much wider variety of languages later in the project, the consortium partners were asked to provide data in their languages. For all the cases, we suggested the following format:

- Text files in UTF-8 format.
- One sentence, number or acronym per line.
- Since the data are a parallel corpus, there are two files. The first one (with .so extension) contains sentences in the source language (original text), and the second one (with .ta extension) contains sentences in the target language (normalized text).

When the system is used by naive users, they will need to create these files for the appropriate language. Expanding un-normalised text out into a string of words is not very difficult, and we do not anticipate any problem with this step. Instructions can be simply in the form of example files for some widely-understood languages such as Spanish and English.

2.6.1.1 Numbers

The first dataset should consist of numbers. In this case, we suggest two parallel corpora: the first one containing isolated numbers (cf. Table 2.6a) and the second containing sentences that include numbers (cf. Table 2.6b). For isolated numbers, we suggest generating sufficient numbers to covering all possible combinations (all ten digits in all possible positions). Depending on the language, the quantity required for acceptable performance might vary. We suggest that 2000 sentences be considered a minimum requirement.

Source	Target
1.234,45	Mil doscientos treinta y cuatro con cuarenta y cinco
1.000.023	Un milln veintitrs
456.234	Cuatrocientos cincuenta y seis doscientos treinta y cuatro

Table 2.6a: Examples of isolated numbers in Spanish

Source	Target
Dame 100 euros	Dame cien euros
Salen 456 toneladas de petrleo	Salen cuatrocientas cincuenta y seis toneladas de petrleo
Son 23,5 euros	Son veintitr'es con cinco euros

Table 2.6b: Examples of sentences with numbers in Spanish

In some languages (like Finnish), it will be necessary to divide the number pronunciation (target language) into morphemes since the number of word types would be too large for the statistical machine translation approach to succeed. Because of this, we suggest creating the target language data divided into morphemes at least for languages with rich morphology (such as Finnish). Morphemes belonging to the same word should be separated with dashes, while different words should be separated with blank characters. For example for Finnish:

- 32534756 (source language)
- kymment-kaksi miljoonaa viisi-sataa-kolme-kymment-nelj-tuhatta seitsemn-sataa-viisi-kymment-kuusi (target language)

2.6.1.2 Acronyms

The second set of training data required consists of acronyms. In this case, we suggest three parallel corpora. The first one contains characters that can be part of an acronyms and how they are pronounced (Table 2.6c). The second one consists of isolated acronyms (Table 2.6d) and the third one contains sentences including acronyms (Table 2.6e). For characters, we propose that **all possible characters that can be part of an acronym are included**. For isolated acronyms, we suggest to generate as many as possible (including at least the 1000 most frequent ones). For the number of sentences containing acronyms, we again suggest 2000 as a minimum requirement.

Source	Target
A	B
B	Be
C	Ce

Table 2.6c: Examples of characters in Spanish

Source	Target
ABC	A be ce
ONU	Onu
UGT	U ge te

Table 2.6d: Examples of isolated acronyms in Spanish

2.6.2 First Romanian experiments

2.6.2.1 Experiments with numbers

In this section, some experiments focused on Romanian numbers will be reported and some comments made about the best configuration options required when adapting the translation platform to deal with numbers.

Source	Target
Acuerdo de la ONU	Acuerdo de la onu
5 reivindicaciones de UGT	Cinco reivindicaciones de u ge te
Llamaron al peridico ABC	Llamaron al peridico a be ce

Table 2.6e: Examples of sentences with acronyms in Spanish

Number of examples			Results	
Train	**Validation**	**Test**	**BLEU (%)**	**WER (%)**
399	1007	3999	85.54	8.88
800	1007	3999	94.53	3.32
1007	800	3999	95.13	3.06
3999	800	1007	98.69	0.79

Table 2.6f: First experiments with numbers in Romanian

The experiments were made on data provided by UTCN - about 6000 isolated numbers without dots, divided into training, validation and test sets. The results we see, shown in Table 2.6g, are that performance improves steadily as the training set becomes larger.

Next, in a similar experiment to that on Spanish, a new strategy for coding the numbers has been proposed and evaluated. The idea is to mix the alternatives considered previously. Initially, the number is divided in groups of 3 digits. Every group of 3 digits is coded independently (as in the original proposal). This way we have less variability and more data to train. In order to deal with the problem of distinguishing "mil" and "millones" sets, specific tokens are included to help in the process.

System or experiment	**BLEU (%)**	**WER (%)**
Not dividing in different tokens using commas Example: 123.400,23 -> 123400,23 -> 1_6 2_5 3_4 4_3 0_2 0_1 , 2_-2 3_-1 In this case the codification of decimals digits starts at the end of the token	94.5	3.3
Including additional tokens Example: 123.400,2 -> 123 . 400 , 2 -> 1_3 2_2 3_1 mil 4_3 0_2 0_1 coma 2_1 In this case the codification of decimals digits starts at the end of the token	98.0	1.2

Table 2.6g: Experiments with isolated numbers

In order to analyse the effect of the size of the training set, we performed additional experiments, with the results shown in Table 2.6h. Other results are presented in the remaining tables. Table 2.6i outlines the most relevant tests that have been undertaken using different n-gram word order, alignments and amounts of data used in training, tuning and testing.

An example of number and its translation is:

12.134 *"douasprezece mii o suta treizeci si patru"*(in Romanian)

Considering different amount of training data	BLEU (%)	WER (%)
200 numbers	96.9	1.7
400 numbers	97.4	1.7
800 numbers	98.0	1.2
1000 numbers	98.3	1.1

Table 2.6h: Romanian experiments with different training sets

No.	Training	Tunning	Testing	Alignment	N-gram	BLEU [%]	WER [%]	SER [%]
1	800	1007	4000	default	3	95.36	1.98	19.23
2	800	1007	4000	default	6	95.43	1.80	18.78
3	800	1007	4000	default	8	95.20	2.08	19.93
4	800	1007	4000	tgtosrc	3	95.93	1.73	17.70
5	800	1007	4000	tgtosrc	6	95.99	1.80	18.30
6	800	1007	4000	tgtosrc	8	96.22	1.80	17.20
7	1007	800	4000	default	3	93.69	2.30	21.41
8	1007	800	4000	tgtosrc	3	96.45	1.65	14.70
9	4000	800	1007	tgtosrc	3	99.13	0.41	4.47
10	4000	800	1007	tgtosrc	6	98.96	0.54	5.16
11	4000	800	1007	tgtosrc	8	98.96	0.48	5.26

Table 2.6i: The influence of the size of training data, alignment type and n-gram order on the translation accuracy for the text pronunciation

The system has been trained starting with a small amount of data (800 sentences) which was increased gradually (1007 sentences, then 4000 sentences). These data sets have been successively rotated for training, tunning and testing. Again, unsurprisingly, the amount of training data is important, as it may be seen in line 9 (Table 2.6i). Next, the type of alignment was changed. Smaller error rates were obtained after modifying the alignment method from 'default' (grow-diag-final-and) to 'tgttosrc' (target to source) alignment. The experiments presented in lines 1 and 4 in Table 2.6i show the new alignment setting's impact, lowering the WER from 1.98% to 1.73% , SER 19.23% to 17.7%, and BLEU is increased by about 0.6%. Then, the order of the n-gram target language model was varied. The choice of a 3, 6 or 8 n-gram results in differences in performance, with the 3-gram providing the lowest error rate when combined with "tgttosrc" alignment and a larger amount of training data. Experiment 9 illustrates the most successful combination of the three factors, with a BLEU of 99.13%, WER 0.41% and SER 4.47%. We believe this represents a satisfactory level of performance for text-to-speech.

Table 2.6j gives the results when separating the words into sublexical units, like morphemes. For example, the number:

12.134 is translated now as: "doua-spre-zece mii o suta trei-zeci-si-patru" (in Romanian), compared with the first approach (12.134, as "douasprezece mii o suta treizeci si patru").

Line 9 of Table 2.6j highlights a notably lower word error rate (WER) compared with experiments reported in Table 2.6i. This method, unlike the experiments reported in Table 2.6i, obtains only a small improvement with the increase of amount of data (column BLEU in Table 2.6j), but this is because error rates are already so low even with small amounts of training data – e.g., just 800 sentences for isolated numbers (lines 4-6 in Table 2.6j).

The experiments with Romanian numbers also adopted the new codification proposed in the Spanish experiments: we separated groups of 3 digits with a separator (X3 - thousands, X6 - millions, X9 - billions, depending on the value of the group of digits) as in the following examples of numbers:

No.	Training	Tunning	Testing	Alignment	N-gram	BLEU [%]	WER [%]	SER [%]
1	800	1007	4000	default	3	97.88	0.99	10.38
2	800	1007	4000	default	6	97.91	0.95	10.28
3	800	1007	4000	default	8	97.87	0.99	10.33
4	800	1007	4000	tgtosrc	3	97.23	1.12	14.65
5	800	1007	4000	tgtosrc	6	97.49	1.12	13.30
6	800	1007	4000	tgtosrc	8	97.26	1.10	13.23
7	1007	800	4000	default	3	88.36	4.09	39.13
8	1007	800	4000	tgtosrc	3	97.66	0.92	11.85
9	4000	800	1007	tgtosrc	3	99.17	0.39	6.06
10	4000	800	1007	tgtosrc	6	98.90	0.5	7.05
11	4000	800	1007	tgtosrc	8	98.88	0.4	6.65

Table 2.6j: The influence of the size of training data, alignment type and n-gram order on the translation accuracy for text split into morphemes

No.	Training	Tunning	Testing	Alignment	N-gram	BLEU [%]	WER [%]	SER [%]
1	4000	800	1007	tgtosrc	3	99.65	0.12	2.28

Table 2.6k: The influence of the new codification of tokens on the translation accuracy for normal text

405.101.542.023,301 is coded as *4_3 0_2 5_1 X9 1_3 0_2 1_1 X6 5_3 4_2 2_1 X3 0_3 2_2 3_1 , 3_-1 0_-2 1_-3*.

The results for RO numbers (we considered only the best configuration obtained in previous tests) with the new codification are in Table 2.6k. The results are better than the baseline approach (Table 2.6i): The WER decreased from 0.41% to 0.12% and the SER decreased from 4.47% to 2.28%.

For experiments in which the new codification of the numbers is applied and the associated text is decomposed into morphemes, the results are presented in Table 2.6l for the best configuration: 4000 sentences for training, 800 sentences for tunning and 1007 sentences for testing.. Compared with the translation results where the group of digits are not separated, but text is still decomposed into morphemes (Table 2.6j) for the same training - tuning - testing configuration, in this case the WER increased from 0.39% to 0.66%, and SER increased from 6.06% to 8.34%. There are more tests planned for this case.

2.6.2.2 Experiments with acronyms

The experiments carried out with acronyms were performed with approximately 50 acronyms and 350 abbreviations. For the experimental data set in Romanian, the number of abbreviations (87.5%) is higher than acronyms (12.5%). In Spanish these percentages are more balanced when estimated automatically from one month of data from El Mundo corpus (see Deliverable 1.1).

The set of acronyms were divided into four parts: three of these were used to build the language model, and the remaining part was used for calculating the features of the acronyms (these features will be calculated with the language model generated before). Once this process was completed with all the possible combinations, we used the features calculated from the acronyms and abbreviations in the WEKA toolkit using a decision tree classifier, in order to obtain a way to discriminate between acronym and abbreviations. The features considered in this experiments are the same that those described in 2.5.3.3 plus three new features:

- Average N-gram Loss: For each character, the system computes the difference between the N-gram for computing the probability, and the maximum possible N-gram for this case. This feature is the average along the character sequence.

No.	Training	Tunning	Testing	Alignment	N-gram	BLEU [%]	WER [%]	SER [%]
1	4000	800	1007	tgtosrc	3	98.52	0.66	8.34

Table 2.6l: The influence of the new codification of tokens on the translation accuracy for the normal text separated into morphemes

- Minimum N-gram Loss: the minimum N-gram order difference (between the real and the theoretical maximum order) along the acronym.
- Maximum N-gram Loss: the maximum N-gram order difference.

Algorithm	Accuracy %	Error instances %
Decision Stump	87.6	12.4
FT	88.3	11.7
LMT	88.8	11.2
NBT Tree	89.3	10.7
REP Tree	90.0	10.0
LAD Tree	90.0	10.0
ADT Tree	90.5	9.5
J48	91.1	8.9

Table 2.6m: First experiments with acronyms in Romanian

After applying various forms of tree classifiers available in Weka, the results shown in Table 2.6m were obtained. The best results were obtained with the J48 algorithm, and we show the confusion matrix and ROC curves for this algorithm in Table 2.6n and Figures 2.6a and 2.6b. We ranked the features using *Weka* depending on the gain of information they provide: Table 2.6o.

Class	Abbreviations	Acronyms
Abbreviations	338	14
Acronyms	22	28

Table 2.6n: Confusion Matrix (J48)

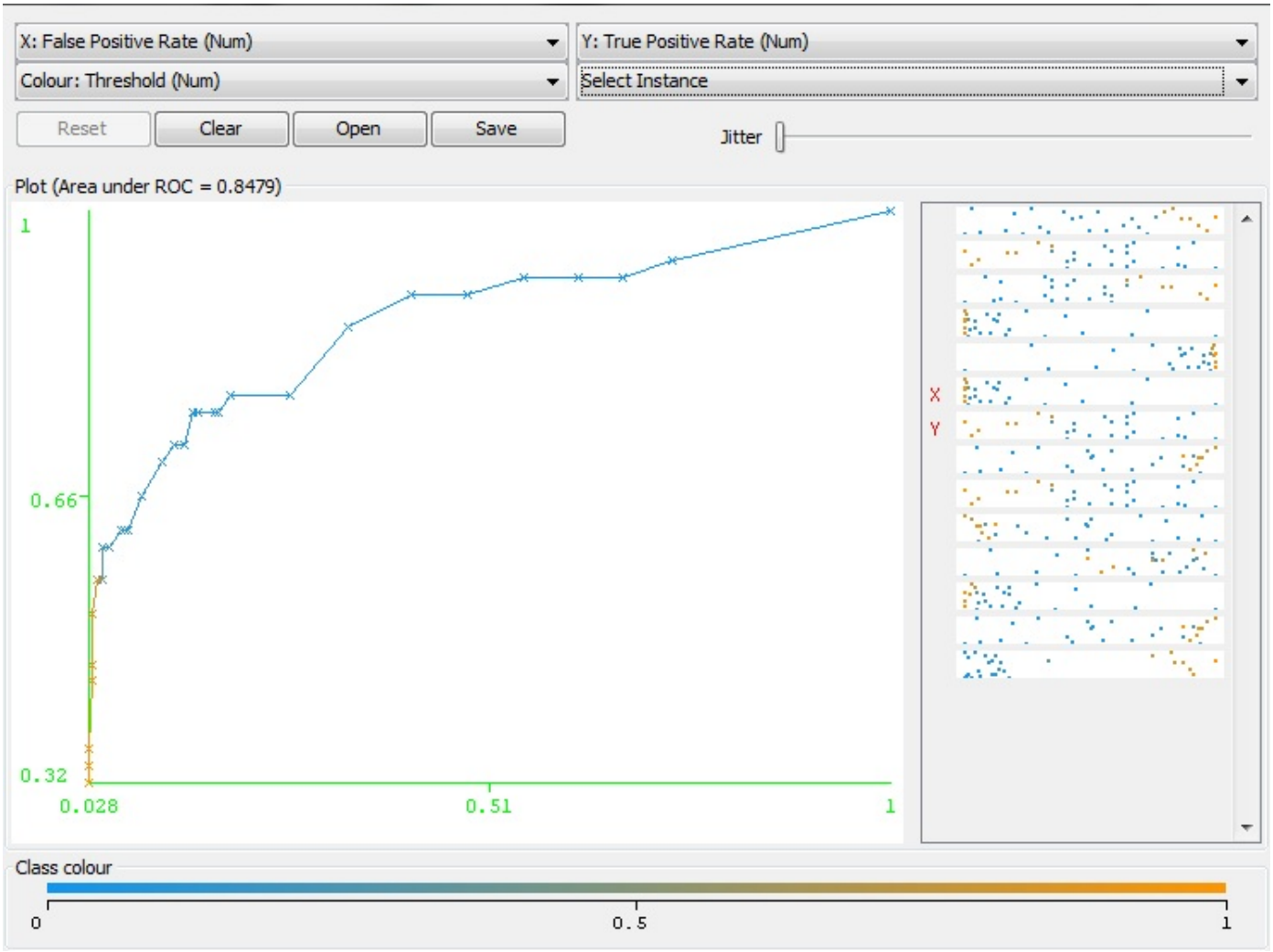

Figure 2.6a: ROC Curve for the pronounceable acronyms

Information Gain	Feature
0.2338	Min. prob.
0.1282	Max. prob
0.0503	Max. ngram loss
0.0429	Perplex.
0.0308	Av. ngram loss
0.0268	Perc. 1gram
0.0249	Av. ngram
0.0236	Min. ngram
0	Min. ngram loss
0	Max. ngram

Table 2.6o: Ranked features

¿¿¿¿¿¿¿ .r153

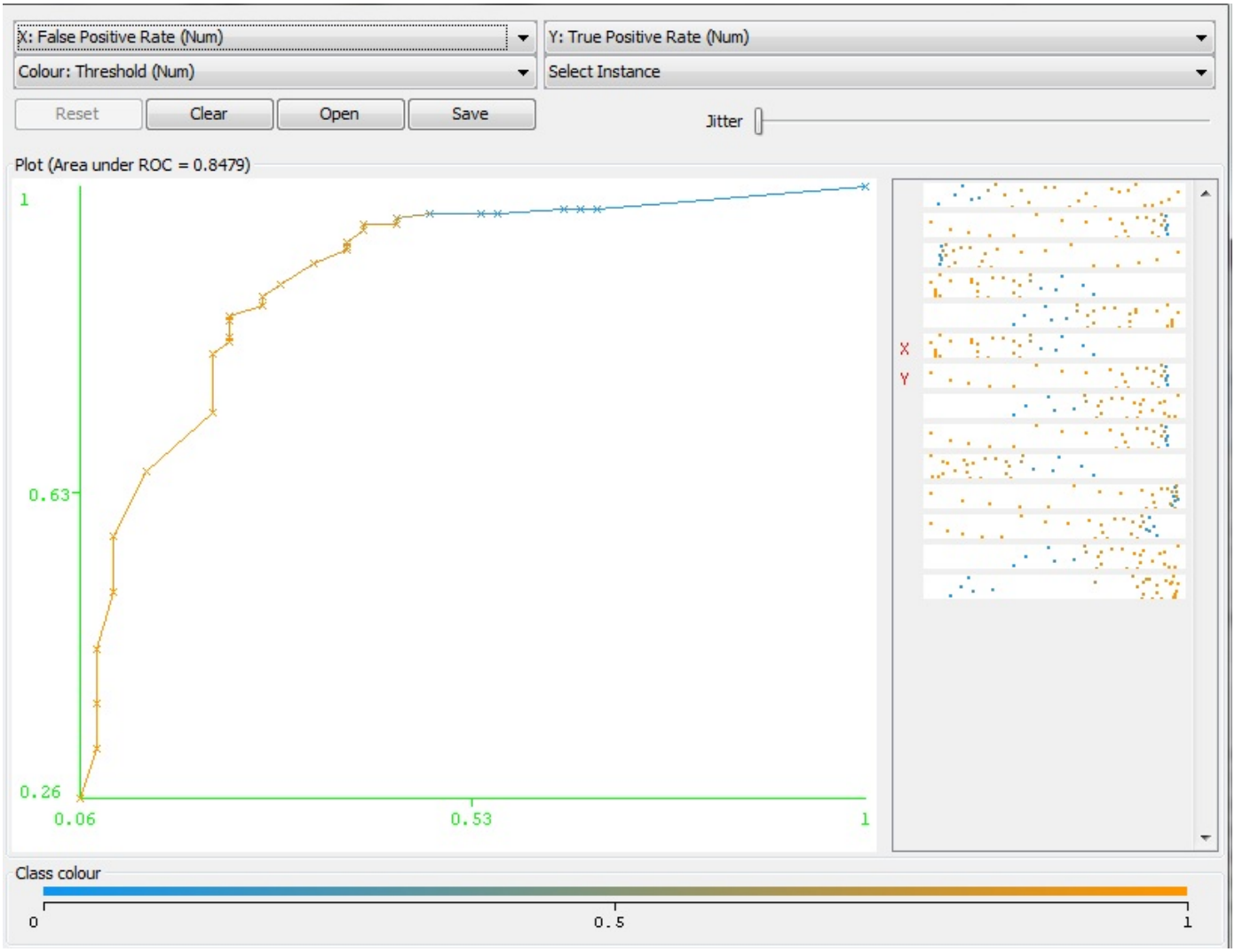

Figure 2.6b: ROC Curve for the non pronounceable acronyms

2.6.3 First English experiments

A small-scale experiment was performed for English, using 6000 isolated numbers without dots. They were divided into sets to train, validate and test the system in varying proportions as showin in Table 2.6p. Once again, larger training sets tend to lead to better performance, and it appears that satisfactory performance can be obtained using approximately this amount of data.

Train	**Validation**	**Test**	**BLEU (%)**	**WER (%)**
400	1000	4000	97.5	1.3
800	1000	4000	98.3	1.0
1000	800	4000	98.5	0.9
4000	800	1000	99.2	0.7

Table 2.6p: First experiments with numbers in English

3 Text analysis

Many of the tools developed for training the text analysis modules implement similar ideas to those outlined in our previous publications [20, 21]. However, those ideas have been re-implemented, and tools implementing new techniques have been added. This part of the deliverable focuses on the practical implementation of these tools in a Python framework that will eventually integrate all front-end processing, including the text normalisation from the previous section and the prosodic analysis that will be introduced in the next section.

3.1 Introduction

The goal of developing these tools is to provide a user-friendly system which is maximally flexible and customisable, and which minimises the time and effort necessary to apply the tools to any new language. The tools have now been used to build working front-ends in a number of languages; however, for these preliminary systems only a limited number of the configurations allowed by the tools have been explored. The tools are designed to allow the rapid implementation and testing of new methods.

The implementation of the tools in SIMPLE4ALL has made use, where possible, of existing general-purpose resources and standards. An example of this is the `Utterance` class which the system uses to represent linguistic/textual structures: this class uses an XML tree to store this structure. XML was chosen with the aim of speeding development as it allows the use of existing open-source XML libraries and parsers.[1] Additionally, the wide use of XML means that users are likely to be familiar with the format of utterance structures even though they are not specialists in text-to-speech: this increases the chance the the tools developed will be usable by a wide audience. Finally, it is hoped that the XML format will encourage integration of the tools with other external software by interested parties, again increasing the usefulness of the tools.

The following sections will describe text analysis modules that can be built with the tools to create a working text analysis system for TTS. The flexibility of the tools means that many different recipes can be used to build a system; the modules described here are largely those of the system built using the tools for the Albayzin 2012 TTS challenge.[2] Consequently, the text of these sections is based heavily on the system description submitted for the challenge, which will be published as [22]. However, considerable extra detail about implementation is included here, to emphasise the ways in which the new tools are flexible and customisable. It should be emphasised that although the system is explained with reference to this Spanish example, there is nothing that is specific to Spanish about the recipe used: it is generally applicable to text ananlysis for TTS in any language making use of an alphabetic script and marking word boundaries orthographically. For example, we have used same recipe for all seven Indian languages of the IIIT-H Indic speech databases[3] to good effect. Finally, while complete TTS systems can be built with these tools, the current toolset should be seen just a baseline example of the use of the framework. The Python framework, use of XML, standardized configuration format and uniform interface to utterance structure provide a solid foundation for further integration and development of more elaborate methods, from unsupervised extraction of relevant phonological and prosodic features, all the way to supervised language-dependent gold standard systems if desired.

3.2 Utterances and utterance processors

As already mentioned, the `Utterance` class stores utterance structures in XML trees, and provides methods for operations such as constructing an utterance from input text, loading/saving XML from/to file, visualising utterance structure as a graph etc.

[1] Specifically, the *lxml* Python XML library is used (`http://lxml.de/` – BSD license), which in turn builds on *libxml2* and *libxslt2* (`http://xmlsoft.org/` and `http://xmlsoft.org/XSLT/` – MIT license).

[2] `http://iberspeech2012.ii.uam.es/index.php/call-for-evalproposals-2/speech-synthesis`

[3] `http://speech.iiit.ac.in/index.php/research-svl/69.html`

The heart of a voice is a list of *utterance processors*. All utterance processors have an `apply_to_utt` method, which performs some modification to the XML structure of an utterance object. To synthesise speech, a voice applies each of the processors in its list to an utterance, with the following method:

```
def synth_utterance(self, input_string):
    utt = Utterance(input_string)
    for processor in self.processors:
        processor.apply_to_utt(utt)
    utt.save()
```

Utterance processors add to the utterance structure incrementally; the final processors used at synthesis time synthesise a waveform for an utterance and play it. The following sections outline some of the analysis and enrichment of utterance structure performed by typical utterance processors before this happens.

3.3 Tokenisation

Text input into the text analyser is currently expected to be in fully normalised form, of the sort produced by the tools described in Section 2. Integration of the text normalisation tools into the Python framework remains to be carried out in the near future. A typical first step to analyse normalised text is to chunk it into tokens. All language-specific knowledge used to provide this chunking in the system developed is derived from the Unicode character database. Each input character is assigned a *coarse category* by lookup in the that database. The three coarse categories used are formed by merging Unicode general categories: coarse category *letter* is made by made by combining general categories L, M and N, coarse category *space* maps exactly to general category Z, and all remaining general categories map to a coarse category called *punctuation*.

Tokenisation is performed by placing token delimiters at the ends of contiguous sequences of characters belonging to the coarse category *letter*. The chunks between these delimiters are tokens. For example, a Spanish utterance might be initialised from text as follows:

```
<utt text="Sí, con seguridad."/>
```

After the utterance processor which performs tokenisation has been applied to the utterance, it will look like this:

```
<utt text="Sí, con seguridad."
  <Token token_class="utt_end" safetext="_UTTEND_" text=""/>
  <Token text="Sí" token_class="letter" safetext="s_LATINSMALLLETTERIWITHACUTE_"/>
  <Token text=", " token_class="punc" safetext="_COMMA__SPACE_"/>
  <Token text="con" token_class="letter" safetext="con"/>
          [...]
</utt>
```

As well as adding the text of tokens as children of the utterance, it can be seen here that various other information is added. Each token is assigned a *token class*, from an inventory of three classes which share the names of the coarse categories. Assignment of a class to a token is performed by placing the coarse categories are placed in an order of precedence: letter, punctuation, space. A given token is classified by traversing this sequence of categories from left to right; if all characters in that token belong to the current category or to a category on the left, the token is assigned to that token class.

Another feature that has been attached to tokens in the example above is *safetext*, which enables the use of tools which can only handle ASCII characters and avoids confusion with special symbols such as field delimiters during processing. Again, use of the Unicode standard means that these forms can be constructed automatically, and allows us to avoid the problems of identifying letters from an unknown character stream.

It should be emphasised that the utterance processor for performing tokenisation described here is very general and easily customisable. It makes heavy use of an existing resource that is in theory universal: Unicode. This means that the processor can be applied to arbitrary new scripts with likely success: for example, it has been applied to the seven Indian languages of the IIIT-H Indic speech databases as mentioned above.

However, the processor is easily user-configurable: the regular expression used for tokenisation and the mapping from Unicode general classes to coarse classes can be done without modifying code, through setting configuration options. Furthermore, all characters encountered in the text of the training data are stored in a text file along with their automatically generated coarse categories and safetexts. This allows a user to manually intervene to correct bad categorisation of characters due to text encoding mistakes or due to errors in the Unicode database, and to specify more user-friendly safetexts than the automatically generated ones (see e.g. the long-winded *_LATINSMALLLETTERIWITHACUTE_* for the character *í* in the example above). However, automatically generated categories and safetexts have so far been used successfully with no manual intervention.

3.4 Lexicon and unsupervised phrase boundary detection

A naive 'lexicon' is used, in which the safetexts of letters of 'letter'-class tokens are used directly as the names of speech modelling units, in place of the phonemes of a conventional front-end. This has given good results for languages with transparent alphabetic orthographies such as Romanian, Spanish and Finnish, and can give acceptable results even for languages with less transparent orthographies, such as English [21, 23, 24, 25]. Using the pronunciations provided by this lexicon, a set of labels is initialised for the speech part of the database by iteratively estimating a set of HMMs and using these to force-align the speech with the labels using a procedure based very closely on that described in [26]. Tokens which are assigned by the tokeniser to the *punctuation* and *space* token classes are allowed by the naive lexicon to be pronounced both as a silence symbol (*sil*) or as a non-emitting symbol (*skip*). As well as determining the timing of letter-boundaries, therefore, the forced alignment procedure determines which space and punctuation tokens are realised as a pause, and which are skipped.

The utterance already shown in the example above looks like this after processors associated with the lexicon have been applied to it:

```
<utt text="Sí,␣con␣seguridad."
  <Token token_class="utt_end" safetext="_UTTEND_" text="">
    <Letter modelname="sil"/>
  </Token>
  <Token text="Sí" token_class="letter" safetext="s_LATINSMALLLETTERIWITHACUTE_">
    <Letter modelname="s"/>
    <Letter modelname="_LATINSMALLLETTERIWITHACUTE_"/>
  </Token>
  <Token text=",␣" token_class="punc" safetext="_COMMA__SPACE_">
    <Letter modelname="sil"/>
  </Token>
  <Token text="con" token_class="letter" safetext="con">
    <Letter modelname="c"/>
    <Letter modelname="o"/>
    <Letter modelname="n"/>
  </Token>
          [...]
</utt>
```

3.5 Tagging: Letter-, morph- and word-representations

3.5.1 Vector Space Model-based tagging

The system makes use of no expert-specified categories of letter and word, such as phonetic categories (vowel, nasal, approximant, etc.) and part of speech categories (noun, verb, adjective, etc.). Instead, features that are designed to stand in for such expert knowledge but which are derived fully automatically from the distributional analysis of a text corpus are used. The distributional analysis is conducted via vector space models (VSMs); the VSM was applied in its original formulation as a model for Information Retrieval to the characterisation of

documents. VSMs are applied to TTS in [21], where models are built at various levels of analysis (letter, word and utterance) from large bodies of unlabelled text. To build these models, co-occurrence statistics are gathered in matrix form to produce high-dimensional representations of the distributional behaviour of e.g. word and letter types in the corpus. Lower-dimensional representations are obtained by approximately factorising the matrix of raw co-occurrence counts by the application of slim singular value decomposition. This distributional analysis places textual objects in a continuous-valued space, which is then partitioned during the training of TTS system components such as acoustic models for synthesis or decision trees for pause prediction. For the voices submitted to the Albayzin Challenge, the text corpus used to provide letter and word features was selected to roughly match the domain of the speech corpus: the entire text of Cervantes' *Don Quijote* (c. 400,000 words), and the text of c. 1800 news stories drawn randomly from those published by the *El Mundo* newspaper in 2006 (giving c. 1 million words). For those voices, a VSM of letters was constructed by producing a matrix of counts of immediate left and right co-occurrences of each letter type, and from this matrix a 5-dimensional space was produced to characterise letters. Token co-occurrence was counted with the nearest left and right neighbour tokens which are *not* of the class *space*; co-occurrence was counted with the most frequent 250 tokens in the corpus. A 10-dimensional space was produced to characterise tokens.[4]

The utterance already shown in the example above looks like this after processors for adding word features have been applied to it:

```
<utt text="Sí,␣con␣seguridad."
            [...]
  <Token text="Sí" token_class="letter" safetext="s_LATINSMALLLETTERIWITHACUTE_"
         word_vsm_d1="-0.650222271361"
         word_vsm_d2="-0.496160551984"
         word_vsm_d3="0.218498104075"
         word_vsm_d4="0.390019669926"
         word_vsm_d5="-0.230137647805"
         word_vsm_d6="0.237969378599"
         word_vsm_d7="0.0707729364758"
         word_vsm_d8="0.0592845156536"
         word_vsm_d9="0.0343895210226"
         word_vsm_d10="0.10899576363">
    <Letter modelname="s"/>
    <Letter modelname="_LATINSMALLLETTERIWITHACUTE_"/>
  </Token>
            [...]
</utt>
```

3.5.2 Morphological decomposition using Morfessor

In previous experiments [21], the word VSM as described in the previous section was found to be inadequate for Finnish: the morphological richness of such languages means that frequency counts of the co-occurrence of surface orthographic forms are too sparse even when large text corpora are used. For such languages, VSMs should be built on the morph instead of the token level. Thus, we have recently integrated Morfessor [28] to our framework, which extracts a morpheme-like segmentation of unannotated text in an unsupervised fashion. However, experiments with morph VSMs have not yet been performed. Possible further applications of Morfessor and its more recent variant, Morfessor Categories MAP, will be found in phrase and accent prediction, lemmatization and compound word detection.

[4]The package *Gensim* [27] was used for performing the singular value decomposition needed to obtain these features.

3.6 Pause prediction and phrasing

The system uses a decision tree to predict whether a token of class *space* or *punctuation* is realised as a pause or not. Data for training the tree is produced from the time-aligned transcriptions of the training data. The predictor variables used for tree training are the token class of the token in question (i.e. whether it is punctuation or space) and the VSM features of the tokens preceding and following the token. The annotation of training data is done by detection of silence in the audio during forced alignment as already described. At run-time, the tree's predictions are used.

It should be noted that the pause predictor makes use of a fully general CARTProcessor object, configured in this way:

```
object_class = CARTProcessor.CARTProcessor
processor_name = pause_predictor
config_file = %(VOICE_COMPONENTS)s/pause_predictor.cfg
cart_location = %(VOICE_COMPONENTS)s/pause_CART.cfg
target_nodes = //Token[@token_class='space']|//Token[@token_class='punc']
tree_context_file = %(VOICE_COMPONENTS)s/pause_contexts.txt
output_attribute = silence_predicted
```

Here, `output_attribute` is the response of a decision tree, and `tree_context_file` lists the independent variables as XPATH expressions (see Section 3.7 below). CARTProcessor is configured in this way to add an attribute called `silence_predicted` to some set of `target_nodes` of an utterance when applied to that utterance (or to train a tree to do this when 'train' method is called). However, it is a general processor that can be configured to predict an arbitrary response from arbitrary input features from an utterance (and to train a tree to do this).

Detected or predicted pauses are used as surrogate 'phrasebreaks': they are used to impose phrase structure upon the XML representation of the utterance, so that the structure of the utterance already shown in the examples above is modified as follows:

```
<utt text="Sí,␣con␣seguridad."
            [...]
 <Token token_class="utt_end" safetext="_UTTEND_" text="" [...]>
    <Letter modelname="sil" [...] />
  </Token>
  <Phrase>
    <Token text="Sí" token_class="letter" [...]
            [...]
    </Token>
            [...]
  </Phrase>
            [...]
</utt>
```

This is the most intricate restructuring of an utterance done by any of the processors developed so far, as it involves adding additional nodes between root and leaves rather than simply adding children to leaf nodes. Also, note that not all tokens become children of phrase nodes: the end-of-utterance token in the above example is not counted as belonging to any phrase. When making XML modifications of this sort, it is easy to err and e.g. let nodes get out of document order. However, the code that has been developed for imposing phrase structure over tokens is fully general, and it is foreseen that this will speed the future development of similar processors for imposing e.g. morpheme and syllable structure.

3.7 Rich contexts and label generation

Information extracted from the utterance structures resulting from text processing is used to create a set of rich contexts for each speech unit in the database. The nature of the features that are extracted from an utterance is specified by a list of XPATH expressions. As with the use of XML for the utterance structures, the wide use of XPATH outside TTS means that users are who are not specialists in text-to-speech likely to be familiar with this formalism. For example, features relating to the length of an utterance are expressed as the following XPATHs:

```
length_utterance_in_phrases
   count(ancestor::utt/descendant::Phrase)

length_utterance_in_words
   count(ancestor::utt/descendant::Token[@token_class='letter'])
```

Note that the use of XPATH lends robustness to the system. For example, removing the processor which adds phrase structure from a voice will render the first feature irrelevant (i.e. it will have the value 0 for each context-dependent model), but the second feature will not be harmed, despite the fact that the nodes which must be traversed between the root of an utterance and Token nodes will be altered. This eases the re-use of feature specifications between different configurations of a synthesiser (for example, configurations with and without phrase structure processors).

The features used in systems built with the tools so far (e.g. the Albayzin Challenge entry) include the identity of the letter to be modelled and those of its neighbours (within a 5-letter window), the VSM values of each of those letters, and the distance from and until a word boundary, pause, and utterance boundary. Word VSM features were not included directly in the contexts, but were used by the decision tree for predicting pauses at runtime.

An utterance's rich-contexts are stored in a label file which can be used for acoustic model training (at training time) or from which speech can be synthesised (at run-time). An utterance's XML structure points to the location of this label file once it has been created, as follows:

```
<utt text="Sí, con seguridad."
            lab="%(TEMPDIR)s/lab/synth_output.lab"
            [...]
</utt>
```

4 External evaluation of voices built using the unsupervised front-end

As mentioned above in Section 3, voices built using the new front-end described here were built for the Albayzin 2012 TTS Challenge (`http://iberspeech2012.ii.uam.es/index.php/call-for-evalproposals-2/speech-synthesis`, [22]).

This Challenge focused on the synthesis of speech in a set of emotions for which training data were provided, and the evaluation reflects this. Evaluation was carried out in four parts: identifiability of the emotion, overall quality of synthetic speech, similarity to the natural voice of the speaker, and perceived emotional intensity. The first of these is scored by accuracy of listeners' guesses at the emotion of the synthetic speech, and the others are scored by normalised Mean Opinion Scores; all four scores are combined using an extended F-measure termed *Emotional Performance measure*.

The other set of synthetic voices entered by the consortium – and those entered by the remaining two groups which took part in the Challenge – employ conventional front-ends incorporating expert linguistic knowledge. For example, they all use a phoneme set and letter-to-sound rules . As such, the unsupervised system was not expected to outperform those systems: the goal was rather to match the performance of these systems as closely as possible but without their reliance on expert knowledge. The unsupervised system achieved an Emotional Performance measure of 0.08, lowest of all systems (0.10, 0.14, 0.15, natural speech: 0.30). However, the gap between the unsupervised system and the lowest-scoring supervised one is small. Furthermore, the unsupervised system outperformed all other systems on the speaker similarity task for the emotions *happiness*, *surprise*, and *sadness*. We consider these acceptable results for an initial baseline system built without any expert language-specific knowledge. Further evaluation of the neutral system – including evaluation for intelligibility – is ongoing.

5 Prominence tagging

Producing expressive prosody, including signalling contrast and emphasis, is a difficult task for a synthesizer based on text-based features only, especially if the features are unreliable, as is the case with current unsupervised methods. Therefore, we opt to integrate acoustics-based word prominence tagging to the SIMPLE4ALL framework. Apart from setting the weights for acoustic parameter types, our current method is unsupervised.

5.1 Method

A universal tendency for signalling prominence is assumed. For example, in the case of F0 and energy, the higher the maximum value within the unit and the larger the movement between the current unit and the next, the more prominent the unit is perceived.

However, direct measurement of these features is not adequate; energy is heavily influenced by phonetic details and F0 varies depending on position in the utterance and, with some languages, phonological tone. To solve this problem, we use a normalization method based on HMM-generated features. A simple synthesizer is first trained, with a deliberately limited contextual feature set. This feature set is designed to allow modelling of phonetic details as well as phrase and utterance level phenomena, but treats the units of interest (words and/or syllables) equally. With the simple synthesizer we generate prosodic parameters for the training data, using with original time alignments. We then subtract these generated parameters from the original analyzed parameters from the natural speech. We further apply a detrending method with a long window for the resulting trajectory to account for long term inconsistencies between utterances in terms of energy, speech rate etc. The resulting difference trajectories should then contain only prominence related information: i.e., the differences between a naive HMM-based prediction and the natural data are attributed to the phenomenon of prominence.

The best set of parameters and measurements is still under development, but currently we use F0, energy and duration as acoustic features. From the difference trajectories, we calculate the rise, mean and fall of each feature type in relation to the current syllable. The mean value is used instead of the maximum for robustness. Weights

are set manually to F0=0.5, energy=0.25, duration=0.25. Rise, fall and max have been given equal weight. The weighted sum of these measurements is then calculated and quantised down to 3 or 4 classes, corresponding roughly to unaccented, weakly accented, strongly accented and emphasis.

The system was implemented for English in the GlottHMM entry for the Blizzard Challenge 2012 with promising results [29]. Figures 5.1a, 5.1b, and 5.1c give an example of the described method, on Finnish speech.

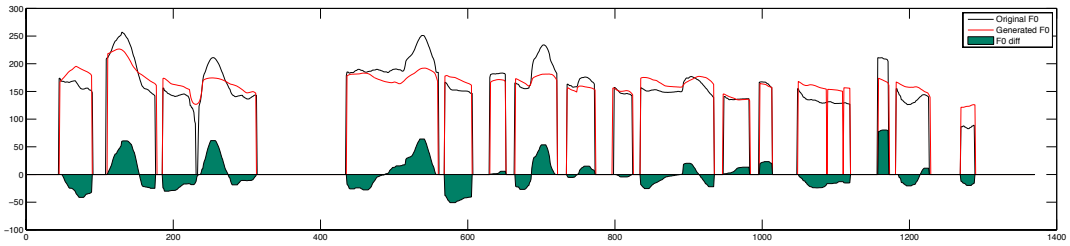

Figure 5.1a: *Example of prominence annotation: Original and generated F0 and their normalized differences*

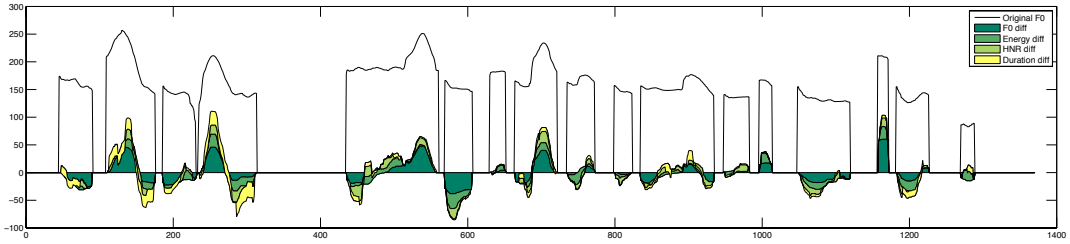

Figure 5.1b: *Example of prominence annotation: Weighted sum of all acoustic features*

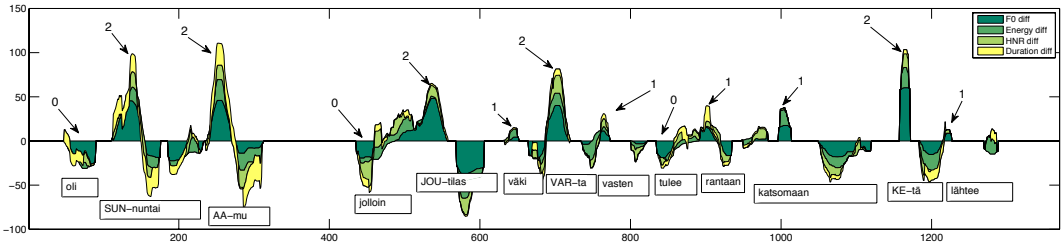

Figure 5.1c: *Example of prominence annotation: Annotated prominence labels on a scale of 0-2*

5.2 Future work

The prominence tagging method has not yet been implemented within the text-analysis framework described earlier, because the prerequisite software infrastucture has only just been finalized. Other future work includes comparing the prominence-tagged synthesis to a text-based word-SVM method, finding proper unsupervised features for prominence prediction and assessing the feasibility of using syllables based on unsupervised syllabification as units of prominence.

6 Conclusions

We have developed all the necessary components for the first version of our front end. A combination of supervised and unsupervised learning from data is used in the various stages. Where supervised learning is used, the data requirements are modest and can easily be met by unskilled end users (i.e., parallel data of unnormalised and normalised numbers). The software framework has been developed and integration of the various components into this framework has now been started. Evaluations of all components have been performed, and performance is satisfactory.

References

[1] R. Sproat, A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards. A normalization of non-standard words. *Computer Speech and Language*, 15:287–333, 2001.

[2] S. Yeates. Automatic extraction of acronyms from text. In *Proceedings of New Zealand Computer Science Research Students' Conference*, pages 167–170, 1999.

[3] Leah Larkey, Paul Ogilvie, Andrew Price, and Brenden Tamilio. Acrophile: An automated acronym extractor and server. In *Proceedings of the ACM Digital Libraries conference*, pages 205–214, 2000.

[4] J.T. Chang, H. Schtze, and Altman. R.B. Creating an online dictionary of abbreviations from medline. *J Am Med Inform Assoc.*, 9:612–620, 2002.

[5] G. Cannon. Abbreviations and acronyms in english word–formation. *American Speech*, 64:99–127, 1989.

[6] Pennell D. and Liu Y. Toward text message normalization: Modeling abbreviation generation. In *Proceedings of the IEEE*, pages 5364–5367, 2011.

[7] R. Sproat. Lightly supervised learning of text normalization: Russian number names. In *Proceedings of the IEEE Workshop on Spoken Language Technology*, pages 436 – 441, 2010.

[8] R. Bikel, D.and Schwartz and R. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34:221–231, 1999.

[9] M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of EMNLP/VLC-99*, pages 100–110, 1999.

[10] D. Bikel, R. Schwartz, and R. Weischedel. Nemo: Extraction and normalization of organization names from pubmed affiliations. *J Biomed Discov Collab*, 5:50–75, 2010.

[11] S. Brody and Diakopoulos. N. Cooooolllllllll!!!!!!!! using word lengthening to detect sentiment in microblogs. In *Proceedings of EMNLP (Conference on Empirical Methods in Natural Language Processing)*, pages 562–570, 2011.

[12] Han B. and T. Baldwin. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of ACL/HLT*, pages 368–378, 2011.

[13] M. Kaufmann. Syntactic normalization of twitter messages. In *Proceedings of ACL/HLT*, pages 368–378, 2010.

[14] A.T. Aw, M. Zhang, J. Xiao, and J. Su. A phrase-based statistical model for sms text normalization. In *Proceedings of ACL*, pages 33–40, 2006.

[15] Pennell D. and Liu. Y. A character-level machine translation approach for normalization of sms abbreviations. In *Proceedings of the IJCNLP*, pages 974–982, 2011.

[16] Och J. and Ney. H. A systematic comparison of various alignment models. *Computational Linguistics*, 29:19–51, 2003.

[17] P. Koehn, F.J. Och, and D Marcu. Statistical phrase-based translation. In *Proceedings of the Human Language Technology Conference (HLT-NAACL)*, pages 48–54, 2003.

[18] A. Stolcke. Srilm - an extensible language modelling toolkit. In *Proceedings of ICSLP*, pages 257–286, 2002.

[19] P. Koehn and H. Hoang. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 868–876, 2007.

[20] Oliver Watts, Junichi Yamagishi, and Simon King. Unsupervised continuous-valued word features for phrase-break prediction without a part-of-speech tagger. In *Proc. Interspeech*, pages 2157–2160, Florence, Italy, August 2011.

[21] Oliver Watts. *Unsupervised Learning for Text-to-Speech Synthesis*. PhD thesis, University of Edinburgh, 2012.

[22] Jaime Lorenzo-Trueba, Oliver Watts, Roberto Barra-Chicote, Junichi Yamagishi, Simon King, and Juan M Montero. Simple4all proposals for the albayzin evaluations in speech synthesis. Accepted at Iberspeech 2012, 2012.

[23] A. Black and A. Font Llitjos. Unit selection without a phoneme set. In *IEEE TTS Workshop 2002*, 2002.

[24] G.K. Anumanchipalli, K. Prahallad, and A.W. Black. Significance of early tagged contextual graphemes in grapheme based speech synthesis and recognition systems. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 4645–4648, 31 2008-April 4 2008.

[25] Matthew P. Aylett, Simon King, and Junichi Yamagishi. Speech synthesis without a phone inventory. In *Interspeech*, pages 2087–2090, 2009.

[26] Robert A. J. Clark, Korin Richmond, and Simon King. Multisyn: Open-domain unit selection for the Festival speech synthesis system. *Speech Communication*, 49(4):317–330, 2007.

[27] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[28] Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*, 4(1), January 2007.

[29] A. Suni, T. Raitio, M. Vainio, and P. Alku. The glotthmm entry for blizzard challenge 2012: Hybrid approach. In *In Proceedings fo the Blizzard Challenge 2012 Workshop*. ISCA, 2012.